Article:

Eberle A. Rambo, Yunsheng Shang, Rolf Ernst, "Providing Integrity in Real-Time Networks-on-Chip", in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.0, no.0.

https://doi.org/10.1109/TVLSI.2019.2906471

Providing Integrity in Real-Time Networks-on-Chip

Eberle A. Rambo, *Member, IEEE,* Yunsheng Shang, and Rolf Ernst, *Fellow, IEEE*

***Abstract*—Mixed-critical real-time systems must meet strict integrity, resilience and timing constraints, as specified by safety standards. Due to the increasing threat of random hardware faults, efficiently achieving high reliability and dependability calls for cross-layer fault-tolerance solutions. This work introduces the Advanced Integrity Q-service (AIQ), a mechanism to ensure the integrity and predictability of on-Chip communication under random hardware faults. Devised for cross-layer and hierarchical fault-tolerance solutions, AIQ realizes low-overhead error detection in hardware and delegates error handling to arbitrary strategies in software. Experimental evaluation featuring benchmark applications and an industrial avionics use case shows that AIQ operates with high reliability and availability and low hardware and performance overheads. In a many-core mixed-critical platform under expected real-time scenarios, AIQ performs with execution time overhead between 1.4% and 7.1%.**

***Index Terms*—Network-on-Chip, integrity, real-time, mixed-criticality, dependability, random hardware faults, soft errors.**

I. Introduction

MODERN Multiprocessor Systems-on-Chip (MPSoCs) architectures now employ Networks-on-Chip (NoCs) as the replacement to the non-scalable bus interconnect. In NoCs, memory operations and I/O are packets traversing a network. On-Chip networks exist in a variety of topologies ranging from meshes, as seen in many-core architectures [1], to topologies optimized for a given MPSoC architecture [2]. Being a central component, the correct operation of the NoC is essential for the MPSoC. An increasingly relevant threat to the correct operation are the random hardware faults [3], which impact the reliability of the computing systems, from large servers to small embedded devices.

NoC-based MPSoC platforms are currently being evaluated for mixed-critical real-time embedded systems [1], [4], [5]. Regulated by safety standards [6]–[8], such systems must meet strict *real-time*, *resilience* and *integrity* requirements. In that context, threats to the intended functionality of the system must be detected and handled appropriately to meet the specified requirements. In case of errors, threats *must* be detected and contained to ensure integrity; a recovery *might* be performed if possible and if resilience is required for reaching high reliability levels; and all *must* be done in a timely and predictable manner under real-time constraints.

The first resilient NoC for real-time systems has been recently proposed in [4], [9] with an approach that ensures the continuous operation of the network after error occurrences. The work is based on results of a Failure Mode and Effects Analysis (FMEA)-based analysis [10] that is capable of meeting certification requirements of safety standards and uncovers all possible impacts of soft errors in the NoC. Upon error occurrence, the approach limits the error impact in time and in scope to provide predictability and integrity. Data delivery under real-time constraints is realized by Automatic Repeat reQuest (ARQ)-based protocols [9], [11]. Although the approach successfully increases the reliability of the NoC, providing error detection and recovery capabilities in hardware incurs overhead even in the absence of errors, which is the case most of the time. Errors in NoCs are seldom and the overhead should be minimized following a "good enough" strategy.

Fast, hardware-based recovery is not always necessary in cross-layer and hierarchical fault-tolerance approaches. Ensuring the system's *integrity* is paramount and seen as basic functionality, e.g., in [12]. In fact, lossless recovery in hardware requires additional circuitry that can incur substantial power consumption and delays – e.g., retransmission buffers in ARQ [11]. Recovery can be performed more efficiently in higher levels of abstraction, as seen in cross-layer approaches with *replicated execution* [13]–[15]. Such techniques exploit the abundant hardware available in multi- and many-core platforms to increase reliability and provide error recovery capabilities in software. Error detection is performed with hardware support, since software-only error detection is ineffective and inefficient. The decision to recover and the error recovery itself are delegated to software.

Nevertheless, the hardware behavior must be predictable since it is a real-time system, and it must detect errors fast enough to allow the system to isolate them and prevent their propagation. That ensures that the recovery, if and as desired, can be carried out in the proper granularity and ensures the integrity of the rest of the system. That reveals two requirements to the hardware operating under soft errors: *integrity* and *real-time* (predictability).

This paper introduces the Advanced Integrity Q-service (AIQ), an end-to-end mechanism to provide integrity and real-time guarantees of NoC transactions under errors. The mechanism is inspired by the idea of keeping track of transactions in distributed systems and Hardware Transactional Memory (HTM). Upon error detection, error handling and recovery are delegated to software, which may react according to an arbitrary strategy in a cross-layer approach. AIQ is proposed and evaluated in a many-core research platform IDAMC [1] considering aspects such as performance and implementation costs. Although the idea of keeping track of transactions in distributed systems is not novel, to the best of our knowledge, its application in hardware in the context of predictable real-time systems has not been explored and evaluated.

The **contribution** of this paper is fivefold: ① the AIQ approach to ensure integrity and predictability of real-time NoCs under random hardware faults; ② formal communication time analysis of AIQ in NoCs with formal analysis for both error-free and error cases; ③ the evaluation of AIQ's performance

Manuscript received July 9th, 2018; revised November 7th, 2018 and February 12th, 2019.

E. A. Rambo, Y. Shang and R. Ernst are with IDA, Braunschweig University of Technology, Hans-Sommer-Str. 66, 38106 Braunschweig, Germany.
E-mail: {rambo, shangy and ernst}@ida.ing.tu-bs.de

This work has been partially funded by the German Research Foundation (DFG) as part of the priority program "Dependable Embedded Systems" (SPP 1500 – spp1500.itec.kit.edu).

https://doi.org/10.24355/dbbs.084-201903190937-0

in a mixed-critical real-time many-core platform, comprising benchmarks and an avionics use case; ④ the evaluation of AIQ's hardware implementation cost; and ⑤ the evaluation of the achieved reliability and availability.

The remaining of the paper is organized as follows. A review of relevant related work is given in Section II. The AIQ approach is introduced in Section III, followed by the formal analysis in Section IV. The experimental evaluation is presented in Section V. Section VI concludes the paper.

II. Related Work

Fault tolerance in NoCs has been widely explored throughout the years. Research has explored fault tolerance both in the link layer [16], [17] and in the network layer [18]–[24]. Moreover, approaches focus on different types of random hardware faults: transient and intermittent faults [16]–[19]; permanent faults [24]; or both [20]–[23]. Comprehensive overviews are found in [25] and [26]. The key technique varies with the approach: from retransmission protocols and adaptive routing to stochastic broadcasts. The authors in [21] tackle transient faults in the link layer with hybrid ARQ/Forward Error Correction (FEC) and permanent faults at the network layer with a reinforcement-learning-based fault-tolerant deflection routing algorithm. A similar approach was adopted in [22]. In contrast, the authors in [23] employ a probabilistic broadcast scheme to reliably transmit packets under transient and permanent faults. In summary, the vast majority of research has focused on general purpose and high performance computing systems and their requirements. Due to different goals and constraints, they are usually not directly applicable to the mixed-critical real-time domain [4].

Mixed-critical real-time systems have strict requirements [6]–[8] that call for dedicated techniques that assure safety without jeopardizing the efficiency of resource usage, the partitioning-sharing trade-off [5]. A comprehensive overview on mixed-criticality is found in [5]. In a mixed-critical real-time NoC, traffic belonging to functions of different criticalities coexist [10] and share resources (routers, links and network interfaces) [5]. Aside from the aforementioned real-time, integrity and resilience requirements, three points are neglected by non-mixed-critical NoCs [4]: predictability and deterministic behavior to enable non-pessimistic, minimum performance guarantees; sufficient independence between different traffic streams to enable the use of the NoC as a shared resource by different traffic of criticalities with independently given performance guarantees; and an error-model that accurately captures all possible impacts of errors, as obtained by FMEAs and usually required by safety standards [6]. The interested reader can refer to [4], [10] for further discussion. Regarding fault-tolerance, beside the aforementioned resilient NoC [4], [9], another work has recently addressed mixed-critical NoCs, albeit considering only permanent faults. Mixed-critical partitioning is employed in [24] to circumvent faulty routers, which are detected by a built-in self-test.

The concept of transactions has been widely applied in different fields of computer science and engineering, such as databases, memories and distributed systems, with the main objective of ensuring the correct execution of transactions while increasing performance and parallelism. In database-management systems [27], transactions are employed to provide the properties of atomicity, consistency, isolation and durability. In that context, the concept is used to increase the level of concurrency, and, thus, performance, in processing numerous simultaneous transactions accessing a single, large database. Thus, transactions might be executed with speculative data accesses and its changes to the database/system are only committed after a validation phase [28]. Speculative means that read and write operations of different transactions are performed concurrently and are not synchronized, which creates a race condition in the event that two transactions access the same data. If an illegal interleaving of reads and writes to the database occurred due to the race condition, the respective transaction is rolled-back and restarted.

Inspired by database-management systems and existing work on HTM [29], the Transactional Memory Coherency and Consistency (TCC) was proposed as an alternative to traditional memory consistency and coherency models [30]. TCC aimed at simplifying parallel software programming and increasing the concurrent performance of shared memory multiprocessors. Unlike traditional consistency models, accesses to critical sections of the code – i.e., lock-based access to shared memory – must not be explicitly specified, but are carried out by transactions, which are atomic from the point of view of consistency. Unlike traditional coherency approaches, data status synchronization can be performed only at the end of a transaction instead of every memory access (the actual operation depends on the coherency scheme). A good overview of transactional memories is given in [31]. In the context of mixed-critical real-time systems, HTM has been explored as hierarchical HTM on distributed embedded systems spanning over on-Chip and off-Chip networks [32].

In the context of fault tolerance, HTM was explored as a hardware-assisted mechanism for error detection and recovery in replicated software execution. The HAFT approach [33] employs instruction-level redundancy for error detection and HTM for recovery with compiler-based code instrumentation. Before committing a transaction to memory, error detection is performed by comparing the instruction-level redundant execution. Recovery is carried out by rolling-back the transaction with HTM. Similarly, FaulTM [34] also employs instruction-level redundancy and HTM but with hardware extensions instead of a software-only approach. Those approaches are types of replicated execution [13], [14]. However, error detection requires *all* tasks/threads in the system to be protected and to execute redundantly. A single unprotected task/thread may cause system failure and violate integrity.

In this paper, the concepts of transactions and ARQ are employed in NoCs to monitor and to detect random hardware faults during runtime and to ensure system integrity while not jeopardizing system performance. The difference with related work lies in that all communication in the NoC is covered instead of only replicated tasks, does not require replicated execution, does not depend on resilient routers, and does not depend on the components of a tile – i.e., whether the tile has a processor with caches, only a hardware accelerator or a memory controller. The hardware ensures that any error in a transaction is detected and, in a cross-layer solution, delegates to software the task of handling the error and choosing the best strategy to do so. Strategies may involve rolling-back, restarting or killing a task/thread, or failing when no other strategy can be safely applied – e.g., [13], [14]. Upon a task failure, the error effects are isolated to ensure the integrity of the rest of the system. Upon a system failure, the failure should be signaled before any erroneous output is performed.

https://doi.org/10.24355/dbbs.084-201903190937-0

III. The AIQ approach

A. Overview

The AIQ approach is a NoC service inspired by memory transactions and ARQ-based protocols. AIQ works as a service that keeps tracks of transactions across the NoC with respect to integrity and timing. Figure 1 gives an overview of how the approach is integrated into the NoC. The AIQ service operates in the transport layer in the NoC and is located between the interfaces of the Network Interface (NI) to the tile internals and the lower layers of NoC. AIQ detects soft and hard hardware errors and reports it to software. The error report signaling path is depicted with the dashed arrows. The AIQ approach delegates to software the task of deciding and recovering from an error in favor of less hardware overhead: Since hardware-based error recovery is not required, power-hungry retransmission buffers and error recovery circuitry are avoided.

B. Transactions in the NoC

The mechanism keeps track of transactions in the NoC in principle by acknowledging successful transactions. However, given the stringent power and area constraints of small-scale on-Chip networks, a closer look at how transactions take place over the NoC is required.

A regular, unprotected transaction is usually initiated by a *master* who sends a request to a *slave*, which will receive the request and respond after it has been completed. For example, tile $t1$ synchronizes with $t2$ by polling the value of a shared variable in the local memory of $t2$. Figure 2a illustrates such a read transaction, where ① $t1$ sends a read request packet and waits for the response. ② $t2$'s NI receives the request packet and forwards the request to the tile. After some time, depending on the resource contention in the tile, ③ the response is packed and sent. The response packet is then received by $t1$ ④, which resumes its execution. Notice that the lower layers of the NoC protocol stack are abstracted away for the sake of clarity. Notice also that the same pattern occurs in case of a write operation or in case of message passing, including or not a response, depending on the memory or communication models.

The *master* might present overlapping memory transactions, e.g., on memory consistency models where support for non-blocking reads and writes exist. The *master* might also support Direct Memory Access (DMA) transfers, which will overlap with regular memory transactions when appropriately used.

A *slave* will potentially receive multiple concurrent requests independently of the memory model since more than one *master* can send a request concurrently. The *slave* can then support single or overlapping requests. In the former case, one request is received and processed at a time, subsequent requests are not accepted by the slave until the completion of the current one. In the latter case, more than one request can be processed at a time. Potentially, in both cases arriving requests will queue up in the NoC causing backpressure and head-of-line blocking, impacting the arrival of subsequent requests and responses. Resource sharing management is a technique that allows the predictable management of shared resources, as might be required in the *slave*. However, that will not be further discussed here, and the interested reader is referred to [35] for more details. For the sake of simplicity and without loss of generality, the discussion in this work assumes single transaction support in *slave*s, unless stated otherwise.

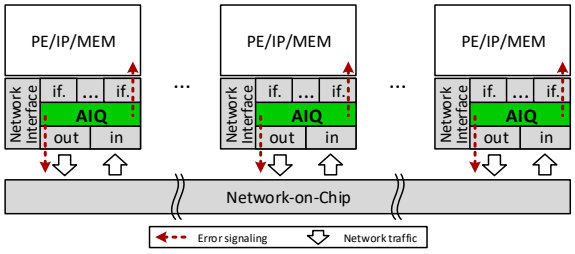


Fig. 1. Overview of the AIQ as a NoC service.

C. Error model

The transactions are composed of packets, which can suffer a series of different impacts due to random hardware faults in the NoC. We derived a functional error model capturing all impacts of soft errors on an unprotected real-time NoC and their durations. The error model is based on the comprehensive description given in [10], which is the result of an FMEA-based analysis methodology that uncovers all impacts of soft errors in the NoC, as required by safety standards [6]–[8].

On an end-to-end communication stream, as seen by the NIs, the impact of random hardware faults can be summarized as follows (packet and data are used interchangeably):

1) *Data corruption*
   - *Correct delivery of corrupt data*: packet is delivered corrupted to the correct destination;
   - *Incorrect delivery of data*: correct packet is delivered to the wrong destination;
2) *Data loss*: packet is not delivered;
3) *Delayed data delivery*: packet is delivered with abnormally longer latency.

The error-free case and the case where the error has no noticeable impact are intentionally left out. Notice that errors 2 and 3 also capture the case where the NoC becomes partially blocked/unavailable due to a soft error with static effects. Soft errors with static effects are transient faults that affect the state of the NoC [10] and, unless they are ruled out with a resilient NoC, their effects last until the circuit is reset [4].

Hard errors have the same impacts on an end-to-end basis as the ones listed above but with a different occurrence pattern. While soft errors are transient or intermittent in nature and affect traffic randomly according with a probabilistic distribution, hard errors are permanent and, upon occurrence, affect the traffic continuously. Thus, the difference between

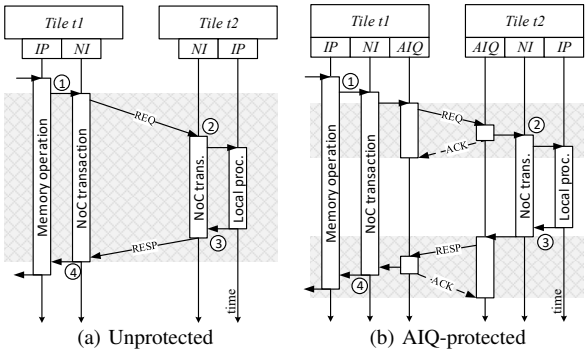


Fig. 2. Example of a transaction over the NoC, consisting of a request (REQ) and a response (RESP).

https://doi.org/10.24355/dbbs.084-201903190937-0

TABLE I
AIQ REQUEST/RESPONSE TRACKING TABLE

Tracking				Diagnosis	
Entry	**State**	**Type**	**Timer**	**Src/Dst**[1]	**Address**
1	Used	Req1	...	0	0xaffd0000
...	...	...	...	...	...
n	Free	...	0	0	0x00000000

soft and hard errors is captured in the error model by the frequency in which a certain error effect occurs. The difference between hard errors and soft errors with static effects is that the latter disappear when the NoC is reset.

D. Protocol

AIQ relies on two well known error detection mechanisms in computer networks: packet integrity check and packet delivery confirmation. The former is based on Error-Detecting Codes (EDCs) and Error-Correcting Codes (ECCs) and can be realized in hardware, or in some cases delegated to software. The latter is realized in hardware with watchdogs and acknowledgment messages, as seen in ARQ-based protocols.

AIQ aims at decoupling the data transport over the NoC from the processing in the tiles, as depicted in the example of Figure 2b. The reason for it is that being predictable and providing timing bounds that include the processing in the tiles creates a complex circular dependency. The traffic in the network depends on the processing in the tiles and vice versa. The AIQ approach should be applicable *without previous knowledge* of the tile internals. Thus, AIQ tracks requests and responses independently in the NoC instead of keeping track of the entire transaction. That is vital for achieving low error detection latencies and effectively limiting the error impact. Moreover, that enables the formal performance analysis and guarantees of the approach.

AIQ's objectives are: ① detect all relevant soft and hard random hardware faults in the NoC; ② operate independently of the tiles' contents and operation and of the NoC topology; ③ report detected errors with very low latencies; and ④ minimize the NoC's performance overhead.

AIQ keeps locally a tracking table common for both requests and responses with n entries, as illustrated in Table I. Every request (resp. response) transmitted by the *master* (resp. *slave*) requires an entry in the table. The entry is kept until the sender confirms the successful transmission of the request (resp. response) or until an error affecting the request (resp. response) is detected. Each entry has a sequence number that identifies the request/response; a flag indicating the entry's state; the request/response type – e.g., read response; a timer; the source and destination of the request[1]; and the address of the request/response. Some fields are used for tracking and others enable the diagnosis after error detection.

The AIQ protocol is divided in two parts: a *master* instance for sending requests and a *slave* instance for sending responses. A summary of the different scenarios is given in Table II. They are described individually in the sequel.

1) Requests: AIQ must account for two cases when tracking requests: loss and corruption.

Loss is monitored with a handshaking mechanism, as seen in ARQ protocols. When the request is sent, the timer associated with the request is triggered. When the request is correctly received, the *slave*'s AIQ sends an Acknowledgement (ACK) packet back to the sender. If the ACK is correctly received, the respective timer is stopped and the request is marked as not pending, releasing the respective entry. The scenario is illustrated in Figure 2b. If the ACK is not received, a timeout will trigger the error detection activities. That occurs if the request itself was lost or if the ACK was lost. In the former case, the *slave* is unaware of the failed request and remains unaffected by the error. That is illustrated in Figure 3a. In the latter case, the *slave* received the request and will process it correctly whereas the *master* cannot be certain that the request was received. Thus, in both cases, only the *master* is notified with a hardware interrupt in order to take appropriate action. The *master* must be able to handle the "orphaned" responses.

[1]The source identifies the interface generating the request/response (cf. Figure 1). The destination can be deducted from the accessed address. Alternatively, it can be stored as Virtual Channel (VC)+route.

TABLE II
AIQ ERROR NOTIFICATION SCENARIOS

	REQ		**RESP**	
	loss	**corrupt**	**loss**	**corrupt**
Sender (*master*)	✓	✓	✓	✓
Receiver (*slave*)		✓	OPT	OPT

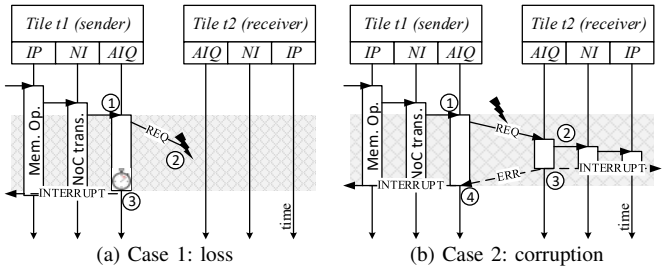


Fig. 3. Illustration of error affecting requests (REQ).

Upon loss detection, four actions are carried out at the *master*:

- the information necessary for error diagnosis is stored in dedicated registers in the AIQ;
- the respective request table entry is released;
- the interface that issued the affected request is notified to abort the transaction in order to allow the further operation of the system;
- and a hardware interrupt is triggered so that appropriate action can be taken in software.

For example, fault containment can be performed by the Real-Time Operating System (RTOS) and error recovery can be performed by a replica manager [13], [14]. Unaffected tasks may continue executing normally.

Corruption is verified with an integrity check using EDC and possibly ECC, both at the *slave* and at the *master*. The integrity check is mandatory for control fields and optional for data[2]. Requests with corrupt control fields are immediately dropped to keep the integrity of the node – e.g., prevents unintended access and modification (corruption) of memory contents. As illustrated in Figure 3b, upon integrity of control fields ② the request is forwarded to the *slave* tile. The integrity check of the data (when enabled) is performed on-the-fly as the data is forwarded to the tile. That improves performance and reduce hardware overhead by avoiding the use of large buffers. The result of the integrity check is available when the last data word of the request traverses the AIQ. Thus, possibly part of the data might have already reached the tile (e.g., memory) by the time the corruption is detected and signaled. Nonetheless,

https://doi.org/10.24355/dbbs.084-201903190937-0

the signaling ③ will occur before the last word of the request leaves the *slave*'s NI. The error is also signaled back to the *master* ④ with a Negative Acknowledgement (NACK) packet.

Upon corruption detection, the following actions are carried out at the *slave*:

- the information necessary for error diagnosis is stored in dedicated registers in the AIQ;
- the interface receiving the affected request is notified to abort the transaction in order to allow the further operation of the system;
- a hardware interrupt is triggered locally so that appropriate action can be taken;
- a NACK is sent to the *master* in order to trigger the error detection actions.

Upon the receipt of a NACK, the following actions are carried out at the *master*:

- the information necessary for error diagnosis is stored in dedicated registers in the AIQ;
- the respective request table entry is released;
- the interface receiving the affected request is notified to abort the transaction in order to allow the further operation of the system;
- a hardware interrupt is triggered locally so that appropriate action can be taken in software.

In case the NACK is not successfully received, e.g. due to the failure of the NoC, the case will be handled as a request loss.

2) Responses: Similarly to requests, AIQ needs to account for two cases when tracking responses: loss and corruption. In both cases, error detection occurs with the same approach and mechanisms as for requests. The difference lies in the error reporting.

The *loss* of a response must be reported back to the *master*. It can be optionally[3] reported locally to the *slave*. Upon the loss detection, the following actions are carried out at the *slave*:

- a NACK is sent to the *master* in order to trigger the error detection actions; the NACK will be transmitted following Stop-and-Wait ARQ.

Optionally, the following actions can be carried out at the *slave* to trigger a reaction locally:

- the information necessary for error diagnosis is stored in dedicated registers in the AIQ;
- a hardware interrupt is triggered locally so that appropriate action can be taken.

Upon the receipt of a NACK, the following actions are carried out at the *master*:

- the information necessary for error diagnosis is stored in dedicated registers in the AIQ;
- the interface receiving the affected request is notified to abort the transaction in order to allow the further operation of the system;
- a hardware interrupt is triggered locally so that appropriate action can be taken in software;

[2]It is optional for data for cases where the written data will be checked for integrity already in software. The data integrity check can be configured during runtime in a memory range granularity, extending the configuration capabilities of the IDAMC platform. The address is considered a control field and its integrity check is mandatory.

[3]The lost response can be used to detect the failure of the NoC due to hard errors or static effects, in which case the *master* will most likely fail to receive the NACK. The tile contents are, however, unaffected by the loss and requires no further action.

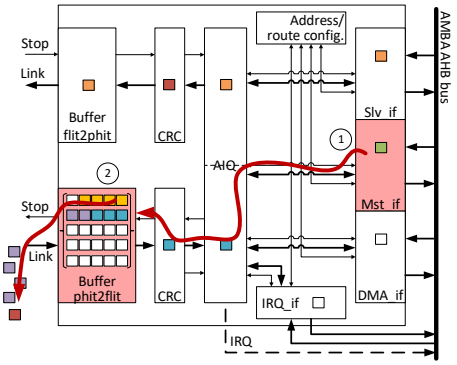


Fig. 4. Block diagram of NI: identified limitations.

- the NACK is acknowledged (Stop-and-Wait ARQ).

The *corruption* of a response is detected and reported locally to the *master*. Upon corruption detection, the following actions are carried out at the *master*:

- the information necessary for error diagnosis is stored in dedicated registers in the AIQ;
- the interface receiving the affected request is notified to abort the transaction in order to allow the further operation of the system;
- a hardware interrupt is triggered locally so that appropriate action can be taken.

E. Discussion and limitations

Due to the size and resource restrictions in a NoC in comparison with an off-Chip network, two points that can impact the performance and correct operation of the approach were identified during design. They are described individually in the sequel along with the applicability limitations of AIQ.

The first identified architectural limitation is related to how received requests are handled by the interfaces of the NI. When handling received requests, the NI handles each request sequentially instead of handling them in parallel or buffering the requests, which would be too costly and unnecessary in most cases. Thus, subsequent requests and packets can potentially be blocked due the backpressure. Figure 4 shows a block diagram of the NI including the AIQ mechanism (corruption detection with CRC is shown separately). The backpressure is illustrated with the red arrow ①. A request is received by the *Mst_if* interface and blocks the following packets as indicated by the arrow. That interface will only accept a next request after the current one has been served. That can make the timing of the approach dependent on the workload and on the internal details of the tile. The key impact on AIQ is that the blocking introduces additional delay to requests/responses, which are expected to be acknowledged as soon as they arrive at the NI. Thus the latency becomes dependent not only on the NoC topology and interfering traffic, but also on the internal performance of the tile. The first point is addressed in this work by bounding the maximum time that an interface in the NI can take to process a request. The bound must be realized by the NI and tile designs – e.g., the NI might abort the request if it is not completed by the tile within the specified bound; alternatively the tile can be designed to accept multiple concurrent requests.

The second identified architectural limitation is related with the NI's input buffer (*buffer phit2flit*), which reassembles flow control units (flits) from physical units (phits) and buffers

https://doi.org/10.24355/dbbs.084-201903190937-0

them until they can be forwarded to the upper layers of the protocol stack. The flits of different VCs are stored in different queues. It can happen that a control message of AIQ (ACK or NACK) experiences head-of-line blocking depending on the arbitration policy and depending on the type of packets queued in front of it. That is illustrated by arrow ② in Figure 4. The situation can escalate to a deadlock, which will be detected as a NoC error, when the *Mst_if* is ready to respond a request but no entries are available in AIQ's tracking table. The ACK that releases a table entry is then blocked in the *buffer phit2flit* due to backpressure from *Mst_if*. The deadlock can be ruled out by using separate VCs for control packets and for requests/responses. Alternatively, it can be ruled out by ensuring that the number of concurrently received requests in a tile is limited and do not block the control packets, e.g. by using a resource manager [35]. The former solution is adopted.

As a mechanism integrated into the NoC, AIQ fails when it is not able to detect and report errors anymore or when the underlying NoC fails. AIQ itself is dimensioned to a single-error scenario, i.e., either the error affects a request/response or it affects the AIQ. For instance, either AIQ's table will suffer a bit flip, which can either be masked or cause the report of an error, or the request/response, handled as expected. The occurrence of a second error in a same request/response is considered a failure.

Finally, notice that AIQ does not support broadcasts and multicasts out-of-the-box since that functionality must be supported by the architecture of the underlying real-time NoC, which is not the case, e.g., in IDAMC to provide sufficient independence to applications [1]. Moreover, AIQ is intended for detecting errors in the NoC. Thus, error detection in tiles and error handling are beyond AIQ's responsibilities. Error handling can be performed in the next levels of a hierarchical fault-tolerance architecture or cross-layer solution. The limitations of such a cross-layer solution depend on factors beyond the scope of the NoC and AIQ, such as the actual software, RTOS, scheduling policies and system-level requirements. The dimensioning and validation of the cross-layer approach can be made with tools such as [13], [36].

Next, the proposed AIQ mechanism is formally analyzed with respect to communication time in the NoC, including the aforementioned aspects.

IV. Formal analysis of AIQ

Transport protocols, such as Go-Back-N ARQ and the proposed AIQ, introduce additional flow control to the communication. That comes from packets that are retained due to handshaking or due to retransmissions with timeouts (in ARQ). It creates a circular dependency where the performance of the transport layer depends on the network latency, which in turn depends on the traffic injected by the transport layer. Similar to [11], we model such networks using Compositional Performance Analysis (CPA) [37], which facilitates the integration of network and transport layer analyses.

The formal timing analysis of AIQ is presented in three parts. First, the modeling in CPA is introduced. Then, the protocol behavior in the error-free scenario in analyzed. Finally, the error case is addressed with an analysis of the worst-case latency in error reporting.

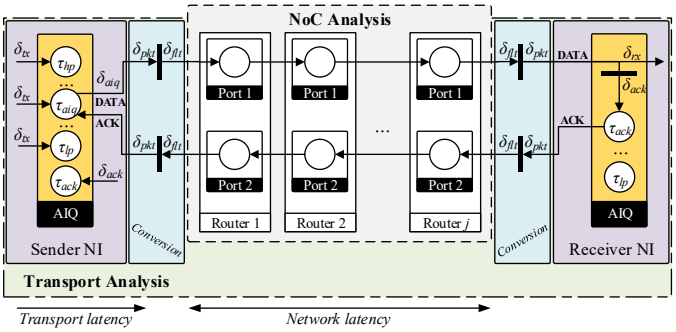


Fig. 5. Modeling of AIQ as a transport layer protocol and the underlying NoC in CPA.

A. Modeling in CPA

CPA [37] relies on independent local analyses of the system resources, such as router ports and CPUs, and a global analysis loop that aggregates the local results to provide Worst-Case Response Times (WCRTs) and jitter of tasks. The system model is based on *resources* providing services, *tasks* consuming these services, and *event models* specifying task activation patterns. Task activations are triggered by an external source or by events propagated from other tasks (predecessor tasks). The activations in an event model are given by event arrival curves $\eta^-(\Delta t)$ and $\eta^+(\Delta t)$, which return the minimum and maximum number of events that can arrive in a given time interval Δt, respectively. Their pseudo-inverse counterparts $\delta^+(q)$ and $\delta^-(q)$ return the maximum and minimum time interval between the first and last events in any sequence of q event arrivals, respectively. A conversion method is presented in [38]. The analysis is then carried out in a local step and global loop. In the local step, the local analysis derives each task's response time and output event model based on the the busy window approach [39]. In a global loop, the analysis propagates tasks' output event models to their dependent tasks, which, in turn, become their input event models. The analysis stops when a fix point is reached and all event models are stable or when predefined constraints are violated, such as a maximum WCRT.

The modeling in CPA is based on the transport layer analysis of [11] and illustrated in Figure 5. An interface in the sender's NI produces packets in a traffic stream according with a given packet-based event model δ_{tx}. The packets are handled at the transport layer by AIQ, which is modeled as a resource. The analysis assumes that packets transmitted from different interfaces within a NI are arbitrated according with Strict Priority Non-Preemptive (SPNP) and also assumes that ACKs and NACKs have the highest priority. Each interface producing packets is modeled as a task τ mapped to that resource – the interface under analysis is depicted as τ_{aiq} whereas a lower priority one and a higher priority one are depicted as τ_{lp} and τ_{hp}, respectively. Packets generated by AIQ are captured by a dedicated task – e.g., ACKs transmitted by the sender NI are captured by τ_{ack}. AIQ then injects traffic into the lower layers of the NoC according to the output event model δ_{aiq}. Notice that only one interface (or AIQ itself) can transmit a packet at a time and therefore only one output traffic stream is depicted for the sender even though several traffic streams can co-exist in the NoC and possibly originate in the same sender.

A protocol with handshaking, such as AIQ, is a bidirectional

https://doi.org/10.24355/dbbs.084-201903190937-0

communication stream [11]. As illustrated in Figure 5, the communication is mapped in the NoC as two unidirectional streams: one for data and one for acknowledgements in the feedback path. As in [11], the underlying NoC analysis is arbitrary. This analysis assumes, without loss of generality, [40] as the underlying NoC analysis, which models the NoC in CPA as follows: each output port of a router is mapped as a resource, and traffic streams are chains of tasks mapped to resources. Resource arbitration depends on the router arbitration. The output of the underlying NoC analysis used by the transport analysis is the worst-case latency L_i^+ of a packet transmitted in a traffic stream i. The interested reader is referred to [11], [40] for further details.

The analysis supports both packet-switched and wormhole-switched NoCs. Packet-switched NoCs are supported by default. For wormhole-switched ones, however, a conversion between event models is necessary, as seen in [11]. That is depicted by the light blue elements in Figure 5. The conversion between packet-based and flit-based event models can be performed with the following equations [11]:

$$\delta^-_{flt,i}(q) = \max\Big\{(q-1)\cdot d_{min}\,,\, \delta^-_{pkt,i}\big(\lceil q \div size_i \rceil\big) + [(q-1) \bmod size_i]\cdot d_{min}\Big\} \tag{1}$$

$$\delta^-_{pkt,i}(q) = \delta^-_{flt,i}\big((q-1)\cdot size_i + 1\big) \tag{2}$$

where $size_i$ is the size of any data packet in stream i (in flits), and d_{min} is the minimum distance between two consecutive flits [40]. For $q \leq 1$, $\delta^-_{flt,i}(q) = 0$ and $\delta^-_{pkt,i}(q) = 0$.

B. Formal analysis: the error-free case

At first sight, the timing behavior of AIQ seems similar to Go-Back-N ARQ [11]. However, AIQ differs from ARQ in that one instance in a NI is shared among traffic streams whereas, in the latter, each traffic stream has its own protocol instance. That makes a big difference. The latter simplifies the analysis by exploiting the fact that all worst-case processing delays and Round-Trip Times (RTTs) are the same for the same traffic stream. In the former, worst-case processing delays and RTTs of interfering packets are potentially different, resulting in the more complex problem of multi-server queues [41], [42]. The analysis of multi-server queues is a hard problem, with a worst-case that is difficult to tightly bound, and it is thus usually handled with Queueing Theory [41]–[43]. Thus, to make the analysis problem feasible, the worst-case analysis of AIQ assumes that there are enough table entries for packets to be transmitted without contention (unlimited number of entries). The adopted strategy allows us to find out the number of entries required to achieve the bounded performance. Similar analysis approaches are seen in the literature [40], [44]. A violation of the assumption can be monitored during runtime and be safely reported.

To obtain the worst-case end-to-end latency of a packet protected by AIQ it is necessary to derive the interference of other traffic in the NI and the contribution of the AIQ protocol to the latency. That is captured by the WCRT of AIQ $R^+_{aiq,i}$, which is the largest period of time in which a packet is retained by the protocol. Similar to [11], the analysis relies on the busy window approach [39]. The first step is to derive the Worst-Case Multiple Packet Queuing Delay.

The Worst-Case Multiple Packet Queuing Delay $Q^+_{aiq,i}(q)$ is the longest time interval from the arrival of the first packet until the q-th packet receives service. The $Q^+_{aiq,i}(q)$ of stream i is given by:

$$Q^+_{aiq,i}(q) = I^+_{lp,i} + I^+_{hp,i}(Q^+_{aiq,i}(q)) + I^+_{acks,i}(Q^+_{aiq,i}(q)) + (q-1)\cdot O^+_{aiq,i} \tag{3}$$

where

$$I^+_{lp,i} = \max_{j\in lp(i)}\{O^+_{aiq,j}\} \tag{4}$$

$$I^+_{hp,i}(\Delta t) = \sum_{j\in hp(i)} \eta^+_{tx,i}(\Delta t)\cdot O^+_{aiq,j} \tag{5}$$

$$I^+_{acks,i}(\Delta t) = \sum_{j\in acks} \eta^+_{tx,i}(\Delta t)\cdot O^+_{aiq,ack} \tag{6}$$

and where $O^+_{aiq,j}$ is the maximum time that AIQ requires to forward a packet of stream j; $O^+_{aiq,ack}$ is the maximum time that AIQ requires to create and forward an ACK; $lp(i)$ and $hp(i)$ are the set of all lower and higher priority streams mapped to the same AIQ as stream i, respectively; and $\eta^+_{tx,i}(\Delta t)$ is the maximum event arrival curve (cf. Section IV-A). Equation 3 results in a fixed-point problem. It can be solved iteratively, starting with a very small, positive ϵ.

Lemma 1. *Equation 3 gives an upper bound on the Worst-Case Multiple Packet Queuing Delay $Q^+_{aiq,i}(q)$.*

Proof. The proof is by induction. When $q = 1$, stream i's packet can be blocked by one non-preemtable, lower priority packet that just started transmitting, assumed to be the largest one causing the longest delay; while queued, the packet can also be blocked by arriving higher priority packets; additionally, the packet can also be blocked by ACKs, which are sent with highest priority, generated due to receiving packets. That is captured in Equation 3 by the terms $I^+_{lp,i}$, $I^+_{hp,i}(Q^+_{aiq,i}(q))$, and $I^+_{acks,i}(Q^+_{aiq,i}(q))$, respectively.

In a subsequent $q+1$-th activation, the packet has to additionally wait for its own previous q packets to be forwarded. That takes $O^+_{aiq,i}$ per packet, potentially increasing the interference caused by higher-priority packets and acknowledgements already captured by $I^+_{hp,i}(Q^+_{aiq,i}(q))$ and $I^+_{acks,i}(Q^+_{aiq,i}(q))$, respectively. That results in Equation 3. □

The Best-Case Multiple Packet Queueing Delay $Q^-_{aiq,i}(q)$ is the shortest time interval from the arrival of the first packet until the q-th packet receives service. It is given by:

$$Q^-_{aiq,i}(q) = (q-1)\cdot O^-_{aiq,i} \tag{7}$$

where $O^-_{aiq,i}$ is the minimum time that AIQ requires to forward a packet of stream i.

Lemma 2. *Equation 3 gives an upper bound on the Best-Case Multiple Packet Queueing Delay $Q^-_{aiq,i}(q)$.*

Proof. The proof is by induction. When $q = 1$, stream i's packet can be forwarded as soon as it arrives. In a subsequent $q+1$-th activation, the packet must wait at most for the previous q packets to be forwarded. That results in Equation 7. □

The Worst-Case Multiple Packet Forwarding Time $B^+_{aiq,i}(q)$ is the longest time interval from the arrival of the first packet until the forwarding of the q-th packet. It extends $Q^+_{iaq,i}(q)$ until the q-th activation completes. The $B^+_{aiq,i}(q)$ of stream i is given by:

$$B^+_{aiq,i}(q) = Q^+_{aiq,i}(q) + O^+_{aiq,i} \tag{8}$$

https://doi.org/10.24355/dbbs.084-201903190937-0

Lemma 3. *The Worst-Case Multiple Packet Forwarding Time $B^+_{aiq,i}(q)$ given by Equation 8 is an upper bound.*

Proof. The proof is by direct deduction. Under SPNP, the time to forward q packets corresponds to the time until the q-th packet is about to receive service and the time it takes to forward the q-th packet, which in non-preemtable. That is captured by the first and second terms of Equation 8, respectively. Both terms are upper bounds and Equation 8 is thus an upper bound. □

The Best-Case Multiple Packet Forwarding Time $B^-_{aiq,i}(q)$ can be similarly derived. It is the shortest time interval from the arrival of the first packet until the q-th packet is forwarded. $B^-_{aiq,i}(q)$ is given by:

$$B^-_{aiq,i}(q) = Q^-_{aiq,i}(q) + O^-_{aiq,i} \tag{9}$$

Lemma 4. *The Best-Case Multiple Packet Forwarding Time $B^-_{aiq,i}(q)$ given by Equation 9 is a lower bound.*

Proof. The proof is omitted. It is similar to Lemma 3, but using lower bounds instead. □

The busy period $w_{aiq,i}$ is the longest time interval in which packets of stream i arrive at AIQ before the previous packet has been transmitted. That is, it is a half-open interval starting with the first activation and ending when activation q completes before the arrival of the $q+1$ activation. The busy period $w_{aiq,i}$ is given by:

$$w_{aiq,i} = \max_{q\geq 1, q\in\mathbb{N}} \left\{ B^+_{aiq,i}(q) \,|\, Q^+_{aiq,i}(q+1) \geq \delta^-_{tx,i}(q+1) \right\} \tag{10}$$

Lemma 5. *The busy window is upper bounded by Equation 10.*

Proof. The proof is by contradiction. Suppose there is a busy window $\breve{w}_{aiq,i}$ longer than $w_{aiq,i}$. In that case, $\breve{w}_{aiq,i}$ must contain at least one activation more than $w_{aiq,i}$, i.e., $\breve{q} \geq q+1$. From Equation 10, $Q^+_{aiq,i}(\breve{q}) < \delta^-_{tx,i}(\breve{q})$, i.e., $\breve{q}$ is not delayed by the previous activation. Since that violates the definition of a busy window, the hypothesis must be rejected. □

The Worst-Case Response Time $R^+_{aiq,i}$ is the longest time interval that any packet of a stream i is delayed by AIQ before being forwarded to the network. It is bounded by:

$$R^+_{aiq,i} = \max_{1\leq q\leq \eta^+_{tx,i}(w_{aiq,i})} \{ B^+_{aiq,i}(q) - \delta^-_{tx,i}(q) \} \tag{11}$$

Theorem 1. *$R^+_{aiq,i}$ (Equation 11) provides an upper bound on the response time of an arbitrary packet in the traffic stream i transmitted under AIQ.*

Proof. The WCRT of an arbitrary packet in the traffic stream i is obtained with the busy window approach [39]. The response time of the q-th packet is the time between its arrival ($\delta^-_{tx,i}(q)$, a lower bound) and its injection in the network ($B^+_{aiq,i}(q)$, an upper bound). The WCRT is then found as the maximum among the response times of activations occurring inside the busy window $w_{aiq,i}$ [39]. It remains to prove that the busy window is correctly captured by Equation 10, and that the blocking captured in Equation 8 is an upper bound. Those are proved in Lemmas 5 and 3, respectively. □

The event model capturing the traffic injection of stream i in the network by AIQ can now be derived. The output event model $\delta^-_{aiq,tx,i}$ propagated by a NI with AIQ is obtained as follows:

$$\delta^-_{aiq,tx,i}(q) = \max\left\{ \delta^-_{tx,i}(q) - R^+_{aiq} + R^-_{aiq}, B^-_{aiq}(q-1) \right\} \tag{12}$$

Theorem 2. *The minimum distance function $\delta^-_{aiq,tx,i}(q)$ given by Equation 12 is a lower bound.*

Proof. Packets can leave AIQ as soon as they arrive but not faster than AIQ is able to process them. That is captured by the $\max$ function. The proof is by cases, with two cases that must be lower bounds. The first case is when the packets leave the AIQ as fast as they are arrive. Since packets can be affected by delay in the AIQ resulting in a jitter ($R^+_{aiq} - R^-_{aiq}$) that is propagated with the output event model. That is guaranteed to be a lower bound. The proof is given in [45]. The second case is that any q packets cannot be closer to each other in time than rate with which AIQ is able to process. That is captured by $B^-_{aiq}(q-1)$, which is proved to be a lower bound in Lemma 4. Since both cases are lower bounds, Equation 12 is also a lower bound. □

The time it takes to transfer q packets can now be bounded, where q might range from a single small packet to a long DMA transfer[4]. The *overall latency* $\mathbb{L}^+_{aiq,i}(q)$ of transmitting q data packets in a stream i is given by:

$$\mathbb{L}^+_{aiq,i}(q) = \delta^-_{tx,i}(q) + L^+_i + R^+_{aiq,i} \tag{13}$$

where L^+_i is the worst-case NoC latency of any packet in stream i, provided by the network analysis (cf. Section IV-A).

Theorem 3. *Equation 13 gives an upper bound on the overall latency to transmit q data packets under AIQ.*

Proof. The proof is by direct deduction. The latency consists of the time it takes for the sender to create q packets ($\delta^-_{tx,i}(q)$), plus the latency it takes for the last (q-th) packet to be delivered by the network, plus the worst-case delay for that packet introduced by AIQ ($R^+_{aiq,i}$) due to contention and handshaking. Due to causality – i.e., packets cannot bypass each other – all previous packets must have been received by the time the last packet is received. Thus, Equation 13 is a valid upper bound. □

Finally, the time it takes to perform a NoC transaction consisting of request and response under AIQ can be bounded. The *transaction latency* $\mathbb{L}^+_{trans}(q_{req}, q_{resp})$ of a transfer comprising q_{req} request packets and q_{resp} is given by:

$$\mathbb{L}^+_{trans}(q_{req}, q_{resp}) = \mathbb{L}^+_{aiq,req}(q_{req}) + O^+_{proc} + \mathbb{L}^+_{aiq,resp}(q_{resp}) \tag{14}$$

where the request and the response consist of q_{req} and q_{resp} packets, respectively, and O^+_{proc} is an upper bound of the time it takes for the transaction to be processed and responded (see Section III-E).

Theorem 4. *Equation 14 gives an upper bound on the overall latency to complete a transaction.*

Proof. The proof is by direct deduction. The latency of a transaction consists of the latency to transmit the request, the time it takes for the receiver to process the request and generate a response, and the latency to transmit the response. That is

[4]DMA transfers can consist of a single, very long packet or multiple smaller packets depending on the NoC architecture and configuration

https://doi.org/10.24355/dbbs.084-201903190937-0

captured by the first, second and third terms of Equation 14. From Theorem 3, the first and third terms are upper bounds. The second term (O^+_{proc}) is an upper bound by definition. Thus, Equation 14 is a valid upper bound. □

C. Formal analysis: the error case

In this section, AIQ is analyzed with respect to its error detection latency guarantees. In contrast to ARQ-based protocols, which guarantee packet delivery, AIQ does not provide error recovery and thus does not introduce itself additional latency due to errors. Error recovery might be performed in software and will certainly incur additional processing time, whose worst-case behavior under errors has been analyzed, e.g., by [13]. As summarized in Table II, two cases must be detected by AIQ – loss and corruption. Upon detection, the error must be notified to the local tile or to the remote tile depending on whether the the affected packet was a request or a response.

The worst-case impact of an error on the detection latency is when the error causes a request/response loss, where the detection of a packet loss occurs upon the timeout event of a timer. In contrast with the detection of corruption, which occurs at the arrival of a request/response, the detection of packet loss will always take longer due to the timeout. Such worst-case error impact is similarly seen in ARQ-based protocols [11]. In the sequel, the worst-case detection latency is analyzed with respect to transient faults. The impacts of permanent faults and permanent effects is discussed afterwards.

In case of request loss, only the local tile must be notified (cf. Table II). The Worst-Case Error Detection Latency for local reporting $\mathbb{L}^{+err}_{aiq,i}(q)$ is the longest time interval between the transmission of a request on stream i until the notification that its transmission on the NoC failed. It is given by:

$$\mathbb{L}^{+err}_{aiq,i}(q) = \delta^-_{tx,i}(q) + R^+_{aiq,i} + t_{out,i} + O^+_{aiq,int} \qquad (15)$$

where $t_{out,i}$ is the timeout value for the request packet of stream i and $O^+_{aiq,int}$ is the maximum delay from timeout detection until a hardware interrupt is raised. Similar to ARQ protocols, the timeout must be chosen larger than the worst-case RTT, usually including a safety margin – i.e., $t_{out,i} > RTT^+_i$.

In case of a response, a remote notification with a NACK is required, which extends the notification latency. That is captured by the Worst-Case Error Detection Latency for remote reporting $\mathbb{L}^{+err\,rem}_{aiq,i}(q)$ and is given by:

$$\mathbb{L}^{+err\,rem}_{aiq,i}(q) = \delta^-_{tx,i}(q) + R^+_{aiq,i} + t_{out,i} + L^+_{nack} + O^+_{aiq,int} \qquad (16)$$

where $t_{out,i}$ is the timeout value for the response packet of stream i, and L^+_{nack} is the worst-case NoC latency of the NACK packet (cf. Section IV-A).

It is possible that the NACK is delivered to the *master* only after retransmission attempts, which can occur in *multiple error scenarios* in very high error rates. In that case, $k \cdot t_{out,NACK}$ can be appended to Equation 16 to account for the k additional retransmissions with timeout $t_{out,NACK}$.

In case of permanent faults or transient faults with static effects causing the failure of the NoC, it is possible that the NACK is not delivered at all. AIQ is also able to detect NoC failures by monitoring the frequency of error occurrences and by detecting the failure of a remote notification. In case of network failure, a dedicated error single-wire signal, shared among all nodes, can be employed to notify the failure to the otherwise unreachable system controller. The controller can then reset the NoC, which will cause the unavailability of the NoC for some time, called Mean Down Time (MDT), whose length depends on the hardware/software implementation. Moreover, the reset of the NoC must be carried out in such a way that the remaining transactions are allowed to finish, so that only the tasks whose transactions failed due to an error will trigger recovery in software. Otherwise, a the reset could induce the failure of all pending transactions and lead therewith to an undesirable scenario.

V. Experimental evaluation

AIQ has been evaluated with respect to performance, implementation overhead, and achieved reliability and availability. The objective of the experiments is to evaluate the impact of AIQ on the regular performance of the MPSoC. AIQ's impact on performance under errors (single-error scenario) is upper bounded by its impact on regular operation (cf. Section IV-C). Notice that a clear distinction must be made between the impact of AIQ and the impact of software execution in the occurrence of errors – e.g., error handling routines. Only the former is evaluated here. On the impact of errors on NoCs, the interested reader can refer to [4], [10].

The performance was evaluated with the many-core platform IDAMC [1]. Benchmark applications as well as an avionics use case were executed on two versions of the platform: a baseline version; and a version with AIQ. Moreover, two different mapping configurations are used to stimulate an extreme scenario and one expected scenario. In the first scenario, the applications are executed remotely inducing the direct impact of the NoC latencies on the application performance – i.e., the application code and data are mapped to memory in remote tiles. In the second scenario, the application nodes execute locally, emulating a Logical Execution Time execution model with inter-core communication for synchronization and DMA for data transfers – i.e., code and data mapped to local memory, and shared memory communication and code download from memory in remote tiles. The two scenarios provide a valuable contrast between AIQ's impact on the NoC traffic and the impact of the NoC performance on the application's execution. Finally, the hardware implementation overhead of AIQ is evaluated, followed by a reliability assessment and discussion. The results presented in this section regard a VHDL design of the IDAMC platform and AIQ simulated in Register-Transfer Level (RTL) with QuestaSim [46] and synthesized as an Application-Specific Integrated Circuit (ASIC) with Design Compiler [47].

A. Performance evaluation: benchmark applications

Let us start by evaluating the performance impact of AIQ with CHSTONE benchmarks [48]. The benchmark applications were mapped to the IDAMC platform as depicted in Figure 6. The applications' code and data were mapped to a remote memory, according with the first of the aforementioned mapping configurations. In the setup, application tiles generate traffic due to cache misses and evictions and due to uncached data access. The applications were divided in two groups: one group (light gray) accesses the memory in tile *DRAM1* and the other group (dark gray) accesses the memory in tile *DRAM2*. The NoC is configured with XY routing for requests, YX

https://doi.org/10.24355/dbbs.084-201903190937-0

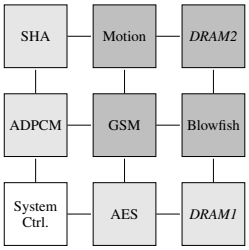


Fig. 6. Mapping of the CHSTONE benchmark applications to the 3x3 NoC.

routing for responses, and a separate VC for each application. The interested reader can refer to [1] and [4], [40] for more details on the IDAMC and on the NoC, respectively.

The results of the RTL simulations can be seen in Figure 7, which plots the latencies of NoC transactions of the different applications as boxplots. Furthermore, the plot compares latencies on an unprotected NoC (base) with latencies on a NoC protected with AIQ. In a boxplot, the whiskers represent the maximum and minimum, the box represents the second and third quartiles, the horizontal line indicates the median value, and the marker indicates the mean value. First, the minimum latency increased 9 cycles in all applications. That is due to the increase in the pipeline length in the NI as seen in Figure 4. By introducing AIQ and the Cyclic Redundancy Check (CRC) checker, the pipeline was extended by four stages for a request and for a response. On average, the latencies increased 16.1% across all applications. The standard deviation increased from 0.42 cycles (AES) up to 2.90 cycles (Blowfish). That is caused by the increased pipeline length as well as the additional feedback traffic consisting of ACKs. This experimental setup intentionally induces a high amount of traffic whose performance strongly impacts the execution time of the applications. Thus, execution time increase varied from 10.8% (*ADPCM*) up to 15.5% (*Motion*), depending on the application's memory footprint. On average, the execution times increased 12.6% across all applications.

Detailed results of the *SHA* application are shown in Figure 8. The figure plots the trace of latencies of NoC transactions in time, comparing the cases with the unprotected and the protected NoC. In the protected NoC, the latencies are slightly longer and show a higher variance than in the unprotected one due to the additional feedback traffic introduced by AIQ. In fact, the average latency increases in about 14.9% from 94.94 to 109.11 clock cycles (cf. Figure 7). Due to high impact of the NoC latencies on application performance artificially induced by the experimental setup, the execution time of *SHA* increased 12.66%. That can be considered as upper bound for the overhead caused by AIQ on cache-enabled executions. As

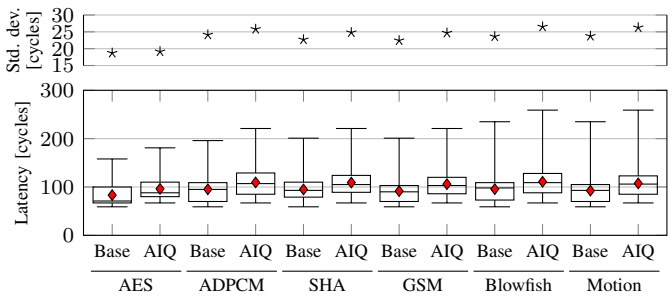
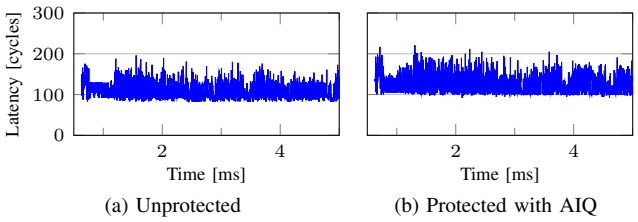


Fig. 7. Observed latency of NoC transactions of CHSTONE benchmarks in a protected (AIQ) and unprotected NoC (Base), 8ns clock period.



(a) Unprotected (b) Protected with AIQ

Fig. 8. Latency of transactions of the *SHA* application executing on an unprotected NoC and on a NoC protected with AIQ, 8ns clock period.

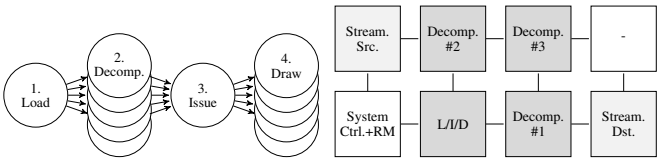
seen next, the impact on performance of executions with local memory is much lower.

B. Performance evaluation: avionics use case

Let us now evaluate AIQ with a parallelized avionics application. Due to the high secrecy involving the development of such systems, the experiments employ an Artificial Demonstration Application (ADA) that mocks the dataflow and workload of a Helicopter Terrain Awareness and Warning System (HTAWS). The original application consists of multiple threads with DAL-C [7] executing on an SMP with an RTOS. The main application dataflow, depicted in Figure 9a, comprises four major pipeline stages. Two of the stages (*Decomp.* and *Draw*) can be parallelized to increase performance by exploiting the available data parallelism. A major frame must be processed from input to output in at most 60ms. In the original single core application, the four stages are executed sequentially, with a period of 60ms. In the parallel version, the stages are executed in a pipeline to increase the overall throughput.

The application is mapped to a 2x4 instance of IDAMC, as depicted in Figure 9b. Stages 1, 3 and 4 are mapped to a single tile (*L/I/D*). Three instances of stage 2 are mapped to different tiles (*Decomp. #1*, *#2* and *#3*). Additionally, interference is introduced in the platform by node *Stream. src*, which generates DMA traffic to node *Stream. dst* with 4.8KB DMA transfers. The system controller (*System Ctrl.+RM*) initializes the platform and manages the access to shared network resources by implementing a resource manager [35].

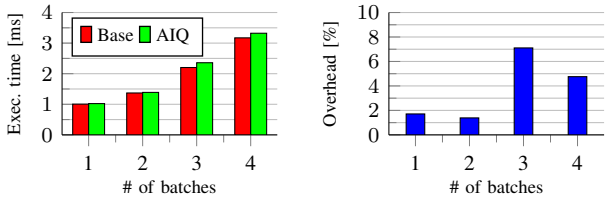
Figure 10 reports the execution time of ADA under different setups in RTL simulations of the IDAMC platform. When increasing the workload, specified as number of batches, ADA takes between 1ms and 3.2ms to execute on IDAMC with an unprotected NoC (Base). With a NoC protected with AIQ, the execution takes from 1.4% to 7.1% longer (cf. Figure 10b). In contrast to the benchmark applications evaluated in the previous section, these executions with AIQ present much lower overhead. That is due to the more realistic setup expected in real-time multi- and many-core platform environments, where a clearer separation between computation and communication is required to limit interference, to achieve predictability



(a) ADA application. (b) Mapping on 2x4 NoC.

Fig. 9. ADA application and its mapping to a 2x4 IDAMC, including interfering applications.

https://doi.org/10.24355/dbbs.084-201903190937-0



(a) Execution time (b) Overhead in time

Fig. 10. Response time comparison of ADA with a protected NoC (AIQ) and an unprotected one (Base).

and to avoid prohibitive analytical over-approximations. That trend can be seen in predictable execution models such as superblocks [49], [50].

Figure 11 reports the latencies of NoC transactions observed in the scenario with 4 batches. The plot shows, as boxplots, the observed latencies of NoC transactions initiated by different ADA application nodes with and without AIQ. The average transaction latencies increase between 14.8% and 19.7% with AIQ. The minimum latencies increase 15.5% due to the additional cycles required by the extended pipeline in the NIs. The observed maximum, however, slightly decreased from 3.9% to 7.8% due to slightly different network traffic patterns resulting, e.g., from the interaction with the feedback traffic. The standard deviation increased at most 1.91 cycles (L/I/D) and decreased at most 1.33 cycles (Decomp. 2).

Although non-negligible, the observed performance overheads are low in comparison with other software and hardware-based fault-tolerance approaches for safety-critical real-time systems [12], [51]. Evaluating only timing can be a pitfall since the cost of the approach might be hidden. For instance, in case of Dual Modular Redundancy (DMR) or Triple Modular Redundancy (TMR) in processors with lock-step execution [12], the performance overhead is low or negligible, the remaining overhead is hidden in the hardware implementation. That is evaluated in the sequel.

C. Implementation overhead

Let us now evaluate the implementation overhead of AIQ. The evaluation features a baseline NI and NIs protected with different configurations of AIQ. The designs were synthesized with Synopsys Design Compiler [47] in an ASIC design-flow with UMC 65nm cell libraries (high and low threshold voltage, worst-case corner 0.9V 125C) targeting a frequency of 125MHz; and with TSMC 28nm cell libraries (high and standard threshold voltage, worst-case corner 0.72V 125C) targeting a frequency of 1GHz. The evaluation involved a NI configured with four interfaces: *DMA if.* for DMA transfers;

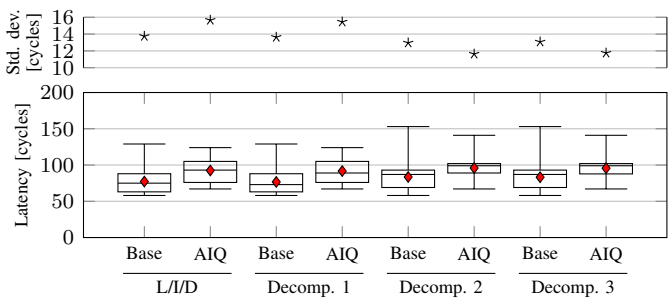


Fig. 11. Observed latency of NoC transactions of ADA with 4 batches in a protected (AIQ) and unprotected NoC (Base), 8ns clock period.

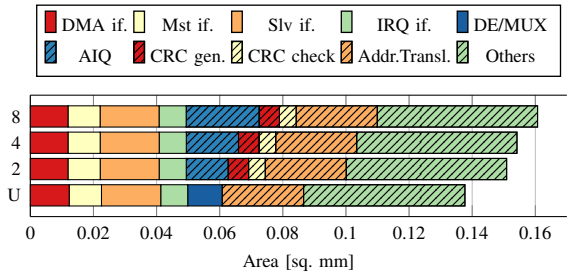


Fig. 12. ASIC synthesis results (65nm UMC) for a NI with AIQ (2, 4 and 8 entries) and without it (U).

Mst if. for remote NoC transactions; *Slv if.* for initiating NoC transactions; and *IRQ if.* for interrupt forwarding.

Figure 12 details the cell area of four different versions of the NI and their internal components: baseline (B) and different configurations of AIQ (2, 4, and 8) where AIQ is equipped with tracking tables with 2, 4 and 8 entries, respectively. In the figures, besides the four aforementioned interfaces, the NI is further divided into mux and demux (*DE/MUX*), which connect the interfaces to the lower layers of the NoC; the lower layers of the NoC are captured by *Others*. The AIQ is subdivided into the core AIQ and the CRC integrity check (*CRC gen* and *CRC check*). The results also show *Addr.Transl.*, which contains the NoC routes and VCs (source routing) as well as local and remote address mapping.

In a 65nm UMC ASIC, a NI implementing AIQ with 2 table entries requires 9.6% additional silicon area. The implementation overhead of AIQ is introduced by the main AIQ component and the CRC integrity check. The main contributors to the area increase are the integrity check (*CRC gen.* and *CRC check*), which correspond for approx. 82.9% of the additional logic. The main component (*AIQ*) contributes with only approx. 17.1% of the additional logic, since it replaces the original multiplexer and demultiplexer (*DE/MUX*). When increasing the number of entries from 2 to 4, 2.2% additional logic is required (in total, 12.0% additional logic from U to 4). Further doubling the number of entries from 4 to 8 requires 4.2% additional logic (in total, 16.7% additional logic from U to 8). Nonetheless, the area requirements of AIQ are expected to decrease with more efficient designs and implementations of the NI and of AIQ itself.

The energy consumption was evaluated with Synopsys PrimeTime [52] using the 65nm netlist. Under full load (single-word memory accesses with random payload), the NI equipped with AIQ (2 entries) consumes 8.03% more energy than baseline. For a larger AIQ with 4 and 8 entries the energy overhead is 9.24% and 11.52%, respectively. When idle (no traffic), the overheads are 6.45%, 7.70% and 9.91% for AIQ with 2, 4 and 8 table entries, respectiely.

Let us now compare the hardware cost of the AIQ approach with the resilient NoC [4] and the DMR and TMR approaches [53]. DMR is able to detect errors of the NoC but it is neither able to pin-point nor to recover from them. As AIQ, DMR can be used to detect errors and achieve integrity. TMR is able to detect errors and also is able to detect which NoC instance is faulty. It can tolerate one error and continue operating.

Figure 13 shows the cost in terms of silicon area of a 5x5 NoC where every router is connected to a NI. Two different technology nodes are used: UMC's 65nm and TSMC's 28nm. Before discussing the results some considerations are required. First, the figure is intended to compare the overhead of

https://doi.org/10.24355/dbbs.084-201903190937-0

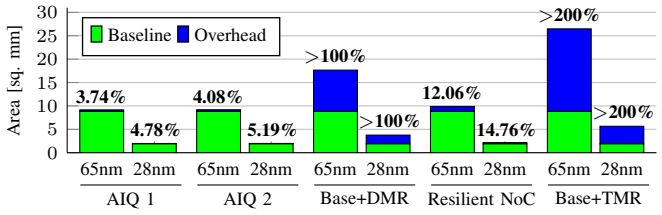


Fig. 13. Comparison of approach overhead for a 5x5 NoC (65nm UMC and 28nm TSMC ASICs).

different approaches in a same technology node. A comparison between the different technology nodes is provided in the sequel (Figure 14). Second, the costs of DMR and TMR do not include the voter and recovery logic, and the Resilient NoC's NIs use a large AIQ as a lower bound for a full-featured ARQ implementation. Third, the results do not account for the link wires, which are highly dependent on the place and routing of the entire MPSoC.

The total cost of implementing AIQ and ensuring the predictability and integrity of a 5x5 NoC is 3.74% in 65nm (4.78% in 28nm) when equipping the NIs with AIQ instances with 2 table entries (AIQ 1). When considering that two NIs are potential bottlenecks and require larger AIQ instances with 8 table entries (AIQ 2), the cost raises slightly to 4.08% in 65nm (5.19% in 28nm). Even when all NIs feature large AIQs, the cost corresponds to 6.52% in 65nm (not plotted). In contrast with DMR and its >100% overhead, AIQ requires just a fraction of the resources to provide the same guarantees. In order to further achieve resilience and high reliability, the resilient NoC approach requires 12.06% additional silicon in 65nm (14.76% in 28nm). In comparison with AIQ, that is roughly three times the overhead. In comparison to TMR, not only is the resilient NoC much more efficient with respect to resource usage (area) and power but also more effective. Besides, DMR and TMR imply a significant increase in the number of interconnecting wires, which can lead to circuit routing complications, potential congestion and lower frequencies during implementation.

A comparison between the two technology nodes can be seen in Figure 14 with the gate equivalent metric, which measures the manufacturing-technology-independent complexity of a digital circuit [54]. The gate equivalent unit corresponds to the design silicon area divided by the area of a reference cell in the same technology node. The usual two-input drive-strength-one NAND cell (NAND2) is employed here. In 28nm, slightly larger overheads can be observed. That comes from the fact that different cells in a cell library – e.g., sequential, combinational, and their different variations – have different sizes. When scaling down the technology node from 65nm to 28nm, the corresponding cells in the library do not scale equally. That results in the case where, e.g., sequential cells

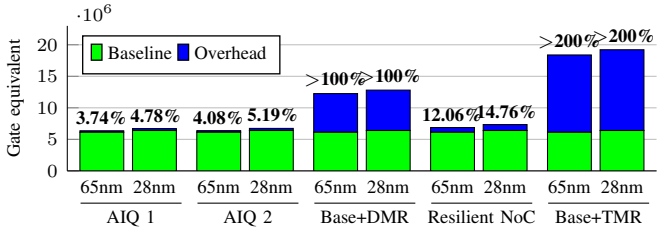


Fig. 14. Gate-equivalent comparison of approach overhead in different ASIC technology nodes (65nm UMC and 28nm TSMC) for a 5x5 NoC.

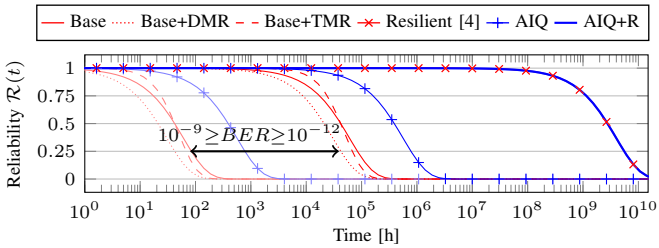


Fig. 15. Reliability comparison of the resilient NoC and non-resilient ones. A 5x5 NoC size is considered.

in the smaller technology node are relatively larger than the combinatorial cells, in comparison with the larger technology node. Thus, although the design is the same, the proportion between different cells (e.g., sequential and combinatorial) in the implementation results in a different area overheads. Those effects can also be observed with the baseline design: notice that the baseline design requires relatively more silicon area (between 4% and 5%) in 28nm than in 65nm.

Regarding maximum achievable frequency, no impact of AIQ has been noticed in 65nm or 28nm – i.e., the critical-path lies in one of the interfaces in all NI configurations.

D. Reliability

To put into perspective what can be achieved with such overheads, let us evaluate the reliability. The reliability is evaluated by means of the reliability metric $\mathcal{R}(t)$, which is the probability that the NoC does not fail during a time interval $[0, t]$ [53]. We reuse the definition of [4], which defines failure in a high dependability mixed-critical real-time system as the violation of integrity, resilience or real-time latency guarantees due to errors, including static effects leading to blocking. Packet loss is not considered as a failure for and AIQ and the Resilient NoC because it is handled in the transport layer. The evaluation considers Bit Error Rates (BERs) from 10^{-12} up to 10^{-9}/hour, which accounts for the BERs expected[5] in practice and the design safety margin [59]. Additionally, a permanent fault rate[6] of 10^{-11}/h per router is considered. The occurrence of a permanent fault leads directly to failure.

Figure 15 plots the analytical $\mathcal{R}(t)$ for the unprotected, baseline NoC and the NoC protected with AIQ considering a 5x5 2D-mesh topology, excluding the NIs. The plot also includes the Resilient NoC [4] and variants of the baseline NoC with DMR (Base+DMR) and TMR (Base+TMR). The DMR and TMR have ideal voters and the latter is non-reparable [53]. Although DMR can ensure the integrity of the system, the extra redundant hardware implies higher susceptibility to errors and results in a less reliable NoC than baseline. The same is seen for TMR, whose capability of withstanding one error does not pay off the extra redundant hardware. On the other hand, AIQ captures all violations of integrity and real-time requirements and is able to increase the reliability in about one order of magnitude w.r.t. the baseline NoC. However, there still exists the possibility that a soft error affects the state of the NoC and causes static effects, leading the blocking of the NoC [4], [10], which still limits the reliability in time. That scenario can either be prevented with

[5]BERs derived [55] for sequential and combinational logic with data from [56] for 65nm CMOS SRAM. Masking effects [57] are not taken into account. BERs in smaller geometries, including FinFETs, have been shown to be decreasing [58] and are thus conservatively covered by the derived BERs.

[6]The fault rate per router is derived from processor failure rates in [60].

https://doi.org/10.24355/dbbs.084-201903190937-0

TABLE III
COMPARISON OF MTTFS IN HOURS

	BER/h	3x3	5x5	8x8
Base	10^{-12}	$5.36 \cdot 10^{5}$	$6.07 \cdot 10^{3}$	$8.65 \cdot 10^{3}$
	10^{-9}	$5.36 \cdot 10^{2}$	$6.07 \cdot 10^{1}$	$8.65 \cdot 10^{0}$
AIQ	10^{-12}	$4.96 \cdot 10^{6}$	$5.63 \cdot 10^{5}$	$8.04 \cdot 10^{4}$
	10^{-9}	$4.96 \cdot 10^{3}$	$5.63 \cdot 10^{2}$	$8.04 \cdot 10^{1}$
AIQ+R & Resilient NoC	10^{-12}	$1.11 \cdot 10^{10}$	$4.00 \cdot 10^{9}$	$1.56 \cdot 10^{9}$
	10^{-9}	$1.11 \cdot 10^{10}$	$4.00 \cdot 10^{9}$	$1.56 \cdot 10^{9}$

a Resilient NoC [4] approach or it can be handled by resetting the NoC back to a valid state with a version of AIQ with NoC resetting capabilities (AIQ+R). AIQ+R and Resilient achieve equally high reliability, since all soft errors in the NoC can be detected and handled by both techniques, at which point hard errors then become the limiting factor.

The conclusions above can also be seen in Table III, which reports the Mean Time To Failure (MTTF) in hours for different NoC sizes and BERs. Even in small topologies, non-resilient NoCs present very low MTTFs and consequently very high failure rates. On average, an 8x8 NoC is expected to be struck by a soft error from every 360 days in a regular environment (BER 10^{-12}) up to every 8.65 hours in an aggressive environment (BER 10^{-9}). Most of those errors present only transient effects and will be handled in software. However, some of them present static effects and their recovery requires resetting the NoC's state. Those are seldom and are expected to occur, on average, every 80.4 hours under BER 10^{-9}. The recovery requires cycles of downtime and thus impacts the NoC availability, which is evaluated next.

Figure 16 reports the unavailability of the NoC, as the complement of its availability ($U = 1 - A$) for different sizes and BERs when varying the MDT. Even for very high BERs, with the MDT in the range of microseconds, the NoC still presents very high availability. As the MDT approaches a tenth of a second, the system experiences longer interruptions due to the NoC recovery, which is likely to compromise its timeliness and violate, already at design time, the real-time guarantees. Thus, fast software routines should be used with hardware support to ensure low MDTs and the applicability of the approach. Alternatively, in case the availability is too low due to a combination of long MDTs, high BERs and large NoC sizes, the Resilient NoC approach [4], which ensures the availability of the NoC in hardware, can be employed.

VI. CONCLUSION

In this paper, we presented AIQ, an end-to-end mechanism to provide *integrity* and *real-time* guarantees for NoC transactions under random hardware faults. When integrated, the mechanism results in a low-overhead fault-tolerant NoC capable of detecting errors and ensuring that their effects are contained in time in order to maintain the system's *predictability* and *integrity*. AIQ explores the idea of keeping track of transactions of distributed systems in the context of NoCs for predictable real-time systems. Upon error detection, error handling and recovery are delegated to software, which may react according to an arbitrary strategy, as seen in cross-layer fault-tolerance approaches. The mechanism was evaluated in the many-core research platform IDAMC considering aspects such as performance, implementation costs, reliability and availability. The experimental evaluation features benchmark applications as well as an industrial avionics use case in scenarios for real-time systems. AIQ operates with low hardware overhead and low impact on performance, as seen in an avionics use case with execution times between 1.4% and 7.1% longer.

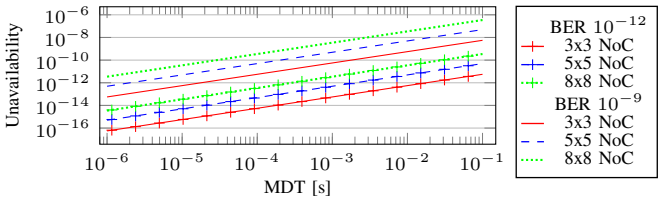


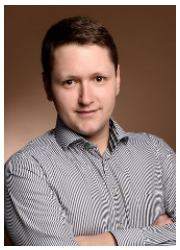
Fig. 16. Unavailability of the NoC with AIQ+R when varying the MDT.

REFERENCES

[1] B. Motruk, J. Diemer, R. Buchty, R. Ernst, and M. Berekovic, "IDAMC: A Many-Core Platform with Run-Time Monitoring for Mixed-Criticality," in *Proc. of HASE'12*, 2012.
[2] J.-J. Lecler and G. Baillieu, "Application driven network-on-chip architecture exploration & refinement for a complex soc," *Design Automation for Embedded Systems*, vol. 15, no. 2, pp. 133–158, Jun 2011.
[3] S. Sayil, *Soft Error Mechanisms, Modeling and Mitigation*. Springer.
[4] E. A. Rambo, C. Seitz, S. Saidi, and R. Ernst, "Bridging the gap between resilient networks-on-chip and real-time systems," *IEEE Transactions on Emerging Topics in Computing*, 2017.
[5] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, p. 82, 2018.
[6] "ISO 26262: Road vehicles – functional safety," International Standards Organization, 2018.
[7] "DO-254: Design assurance guidance for airborne electronic hardware," RTCA Incorporated, 2000.
[8] "IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems, ed.2.0," International Electrotechnical Commission, 2010.
[9] E. A. Rambo, C. Seitz, S. Saidi, and R. Ernst, "Designing networks-on-chip for high assurance real-time systems," in *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, Jan 2017.
[10] E. A. Rambo, A. Tschiene, J. Diemer, L. Ahrendts, and R. Ernst, "FMEA-Based Analysis of a Network-on-Chip for Mixed-Critical Systems," in *2014 Eighth IEEE/ACM International Symposium on Networks-on-Chip (NoCS)*, Sept 2014, pp. 33–40.
[11] E. A. Rambo, S. Saidi, and R. Ernst, "Providing formal latency guarantees for ARQ-based protocols in Networks-on-Chip," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016.
[12] Infineon, "AURIX™ – safety joins performance," https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/aurix-safety-joins-performance/, 2018, [Online].
[13] E. A. Rambo and R. Ernst, "Replica-Aware Co-Scheduling for Mixed-Criticality," in *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, vol. 76, 2017, pp. 20:1–20:20.
[14] P. Axer, R. Ernst, B. Döbel, and H. Härtig, "Designing an analyzable and resilient embedded operating system," in *Proc. on Software-Based Methods for Robust Embedded Systems*, Braunschweig, Germany, 2012.
[15] Y. Zhang and K. Chakrabarty, "Dynamic adaptation for fault tolerance and power management in embedded real-time systems," *ACM Trans. Embed. Comput. Syst.*, vol. 3, no. 2, pp. 336–360, May 2004.
[16] Q. Yu, B. Zhang, Y. Li, and P. Ampadu, "Error control integration scheme for reliable NoC," in *ISCAS*, 2010.
[17] D. Bertozzi, L. Benini, and G. De Micheli, "Error control schemes for on-chip communication links: the energy-reliability tradeoff," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, no. 6, 2005.
[18] J. Kim, D. Park, C. Nicopoulos, N. Vijaykrishnan, and C. Das, "Design and analysis of an NoC architecture from performance, reliability and energy perspective," in *ANCS*, 2005.
[19] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. Das, "Exploring fault-tolerant Network-on-Chip architectures," in *Proc. of DSN'06*, 2006.
[20] A. Kohler, G. Schley, and M. Radetzki, "Fault tolerant network on chip switching with graceful performance degradation," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 29, no. 6, 2010.
[21] C. Feng, Z. Lu, A. Jantsch, M. Zhang, and Z. Xing, "Addressing transient and permanent faults in NoC with efficient fault-tolerant deflection router," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 6, 2013.

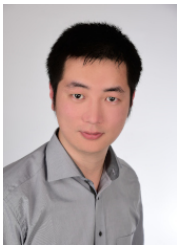
https://doi.org/10.24355/dbbs.084-201903190937-0

[22] C. Killian, C. Tanougast, F. Monteiro, and A. Dandache, "Smart reliable Network-on-Chip," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 22, 2014.
[23] P. Bogdan, T. Dumitraş, and R. Marculescu, "Stochastic communication: A new paradigm for fault-tolerant networks-on-chip," *VLSI design*, vol. 2007, 2007.
[24] T. Hollstein, S. P. Azad, T. Kogge, and B. Niazmand, "Mixed-criticality noc partitioning based on the nocdepend dependability technique," in *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2015 10th International Symposium on*. IEEE, 2015, pp. 1–8.
[25] M. Radetzki, C. Feng, X. Zhao, and A. Jantsch, "Methods for fault tolerance in Networks-on-Chip," *ACM Comput. Surv*, vol. 46, no. 1, pp. 8:1–8:38, 2013.
[26] S. Werner, J. Navaridas, and M. Luján, "A survey on design approaches to circumvent permanent faults in networks-on-chip," *ACM Comput. Surv.*, vol. 48, no. 4, Mar. 2016.
[27] J. Gray and A. Reuter, *Transaction processing: concepts and techniques*. Elsevier, 1992.
[28] H. T. Kung and J. T. Robinson, "On optimistic methods for concurrency control," *ACM Trans. Database Syst.*, vol. 6, no. 2, pp. 213–226, Jun. 1981.
[29] M. Herlihy and J. E. B. Moss, "Transactional memory: Architectural support for lock-free data structures," *SIGARCH Comput. Archit. News*, vol. 21, no. 2, pp. 289–300, May 1993.
[30] L. Hammond, V. Wong, M. Chen, B. D. Carlstrom, J. D. Davis, B. Hertzberg, M. K. Prabhu, H. Wijaya, C. Kozyrakis, and K. Olukotun, "Transactional memory coherence and consistency," *SIGARCH Comput. Archit. News*, vol. 32, no. 2, pp. 102–, Mar. 2004. [Online]. Available: http://doi.acm.org/10.1145/1028176.1006711
[31] T. Harris, A. Cristal, O. S. Unsal, E. Ayguade, F. Gagliardi, B. Smith, and M. Valero, "Transactional memory: An overview," *IEEE Micro*, vol. 27, no. 3, pp. 8–29, May 2007.
[32] Z. Owda, M. Urbina, R. Obermaisser, and M. Abuteir, "Hierarchical transactional memory protocol for distributed mixed-criticality embedded systems," in *Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress, 2016 IEEE 14th Intl C*. IEEE, 2016, pp. 334–343.
[33] D. Kuvaiskii, R. Faqeh, P. Bhatotia, P. Felber, and C. Fetzer, "HAFT: hardware-assisted fault tolerance," in *Proceedings of the Eleventh European Conference on Computer Systems*. ACM, 2016, p. 25.
[34] G. Yalcin, O. Unsal, and A. Cristal, "Faultm: error detection and recovery using hardware transactional memory," in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*. IEEE, 2013.
[35] A. Kostrzewa, S. Saidi, L. Ecco, and R. Ernst, "Dynamic admission control for real-time networks-on-chips," in *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*. IEEE, 2016.
[36] P. Axer, M. Sebastian, and R. Ernst, "Reliability analysis for mpsocs with mixed-critical, hard real-time constraints," in *Proc. Intl. Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2011.
[37] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System Level Performance Analysis–the SymTA/S Approach," *IEE Proceedings-Computers and Digital Techniques*, vol. 152, no. 2, 2005.
[38] J. Diemer, P. Axer, and R. Ernst, "Compositional performance analysis in python with PyCPA," *Proc. of WATERS*, 2012.
[39] K. Tindell, A. Burns, and A. Wellings, "An extendible approach for analyzing fixed priority hard real-time tasks," *Real-Time Systems*, vol. 6, no. 2, 1994.
[40] E. A. Rambo and R. Ernst, "Worst-case communication time analysis of networks-on-chip with shared virtual channels," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015.
[41] P. Vijaya Laxmi and U. Gupta, "Analysis of finite-buffer multi-server queues with group arrivals: Gix/m/c/n," *Queueing Systems*, vol. 36, no. 1, pp. 125–140, Nov 2000.
[42] D. Manfield and P. Tran-Gia, "Analysis of a finite storage system with batch input arising out of message packetization," *IEEE Transactions on Communications*, vol. 30, no. 3, pp. 456–463, Mar 1982.
[43] A. E. Kiasari, A. Jantsch, and Z. Lu, "Mathematical formalisms for performance evaluation of Networks-on-Chip," *ACM Computing Surveys*, vol. 45, no. 3, 2013.
[44] J. Diemer, J. Rox, M. Negrean, S. Stein, and R. Ernst, "Real-Time Communication Analysis for Networks with Two-Stage Arbitration," in *EMSOFT'11*, October 2011.
[45] S. Schliecker, J. Rox, R. Henia, R. Racu, A. Hamann, and R. Ernst, *Formal performance analysis for real-time heterogeneous embedded systems*. CRC Press, 2009, pp. 57–92.
[46] *Questa SIM User's Manual*, Mentor Graphics Corp., Wilsonville, OR, 2016.
[47] *Design Compiler User Guide*, Synopsys Inc., Mountain View, CA, 2017.
[48] Y. Hara, H. Tomiyama, S. Honda, and H. Takada, "Proposal and quantitative analysis of the chstone benchmark program suite for practical c-based high-level synthesis," *Journal of Information Processing*, vol. 17, pp. 242–254, 2009.
[49] A. Schranzhofer, J.-J. Chen, and L. Thiele, "Timing analysis for TDMA arbitration in resource sharing systems," in *Proc. of RTAS'10*, 2010.
[50] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley, "A predictable execution model for COTS-based embedded systems," in *Proc. of RTAS'11*, 2011.
[51] M. Hoffmann, F. Lukas, C. Dietrich, and D. Lohmann, "dOSEK: the design and implementation of a dependability-oriented static embedded kernel," in *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, April 2015, pp. 259–270.
[52] *PrimeTime User Guide*, Synopsys Inc., Mountain View, CA, 2017.
[53] A. Høyland and M. Rausand, *System reliability theory: models and statistical methods*. John Wiley & Sons, 2009, vol. 420.
[54] H. Kaeslin, *Digital integrated circuit design: from VLSI architectures to CMOS fabrication*. Cambridge University Press, 2008.
[55] T. Heijmen, *Soft Errors from Space to Ground: Historical Overview, Empirical Evidence, and Future Trends*. Springer US, 2011, pp. 1–25.
[56] J. Autran, P. Roche, S. Sauze, G. Gasiot, D. Munteanu, P. Loaiza, M. Zampaolo, and J. Borel, "Altitude and underground real-time SER characterization of CMOS 65nm SRAM," *IEEE Transactions on Nuclear Science*, vol. 56, no. 4, 2009.
[57] M. Ebrahimi, A. Evans, M. B. Tahoori, E. Costenaro, D. Alexandrescu, V. Chandra, and R. Seyyedi, "Comprehensive analysis of sequential and combinational soft errors in an embedded processor," *IEEE Trans. on CAD (TCAD)*, vol. 34, no. 10, 2015.
[58] S. Lee, I. Kim, S. Ha, C. s. Yu, J. Noh, S. Pae, and J. Park, "Radiation-induced soft error rate analyses for 14 nm FinFET SRAM devices," in *2015 IEEE International Reliability Physics Symposium*, April 2015.
[59] J. Leray, "Effects of atmospheric neutrons on devices, at sea level and in avionics embedded systems," *Microelectronics Reliability*, vol. 47, no. 9-11, 2007, 18th European Symposium on Reliability of Electron Devices, Failure Physics and Analysis.
[60] F. Oboril and M. B. Tahoori, "Exploiting instruction set encoding for aging-aware microprocessor design," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 21, no. 1, pp. 5:1–5:26, Dec. 2015.



Eberle A. Rambo received a B.S. and M.S. degrees in Computer Science from the Federal University of Santa Catarina, Florianópolis, Brazil, in 2009 and 2011, respectively.

Since 2013, he is pursuing his PhD at Braunschweig University of Technology, Braunschweig, Germany. His research activities include dependable mixed-critical real-time systems and Networks-on-Chip.



Yunsheng Shang received a B.S. degree in Microelectronics from the Xidian University, Xi'an, China, in 2014. He is currently pursuing a M.S. degree in Electronics at Braunschweig University of Technology, Braunschweig, Germany.

Rolf Ernst received a Diploma degree in Computer Science and the Dr.-Ing. degree in Electrical Engineering from the University of Erlangen-Nuremberg, Erlangen, Germany, in 1981 and 1987, respectively. From 1988 to 1989, he was with Bell Laboratories, Allentown, PA.

Since 1990, he has been a professor of Electrical Engineering at Braunschweig University of Technology, Braunschweig, Germany. His research activities include embedded system design and design automation.

https://doi.org/10.24355/dbbs.084-201903190937-0